Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Embedded Systems

Laboratory work #2

User Interaction: STDIO + LCD + Keypad

Author:
Alexandru Cebotari
std. gr. FAF-233

Verified:
Martiniuc Alexei

Chişinău 2026

Theory Background

This laboratory studies user interaction on an embedded system by combining three communication channels: keypad input, LCD output, and serial text I/O through STDIO. The solution targets Arduino Uno (ATmega328P) and uses a modular architecture where each hardware peripheral is encapsulated in dedicated `.h/.cpp` modules.

The practical objective is to implement a keypad-based access control behavior with immediate visual feedback:

- user enters a 4-digit code on a 4x4 keypad;
- LCD 16x2 (parallel 4-bit mode) displays guidance and result messages;
- green LED indicates valid code and red LED indicates invalid code;
- serial terminal displays runtime information using `printf`/`scanf`.

Hardware components and role:

Table 1: Main hardware components

#	Component	Specification	Role
1	Arduino Uno R3	ATmega328P, 16 MHz	Main MCU
2	LCD 16x2	HD44780, 4-bit parallel mode	User messages
3	Keypad 4x4	Matrix keyboard	Numeric input
4	Green LED + resistor	220 Ω	Valid indicator
5	Red LED + resistor	220 Ω	Invalid indicator
6	Breadboard + jumpers	—	Interconnection

Software context:

- VS Code + PlatformIO for project management and build.
- Arduino framework and libraries: `Keypad`, `LiquidCrystal`.
- AVR libc stream binding with `fdev_setup_stream()` for STDIO over UART.

Architectural and HW–SW design overview:

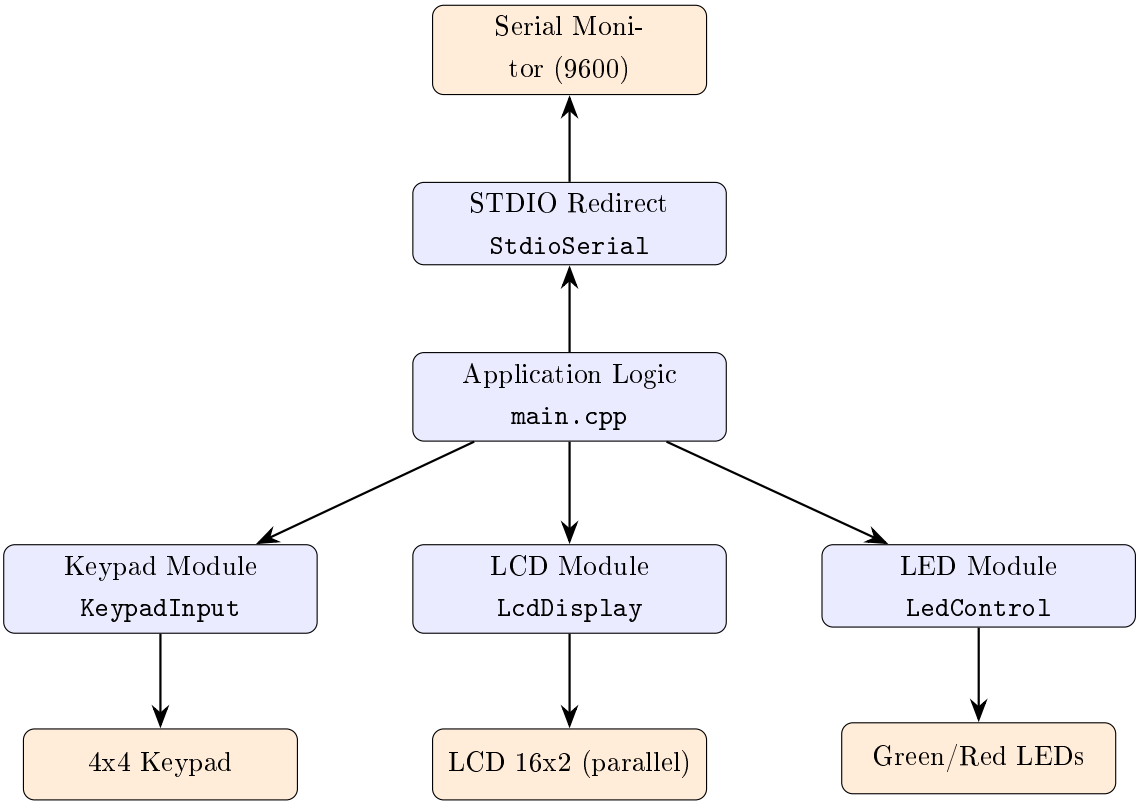

Figure 1: Modular architecture and hardware–software communication

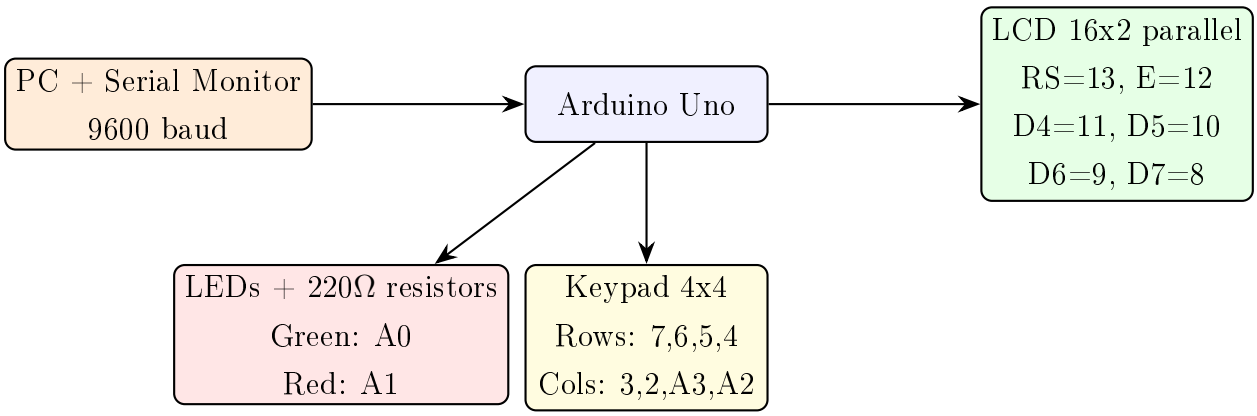

Figure 2: Logical electrical interconnection

The Task

The lab assignment was implemented according to the required behavior:

1. Read a 4-digit code from keypad and evaluate validity.
2. Display prompts and final status on LCD.
3. Signal result via green/red LEDs.

4. Use STDIO services (`printf`, `scanf`) for serial I/O.
5. Keep project modular with separate interfaces and implementations.

Implemented optional extension:

- Runtime password change mode from keypad (documented and justified).

Technical implementation

The source tree follows the required modular organization:

```
Lab2/
+-- Condition.txt
+-- Code/
|   +-- platformio.ini
|   +-- include/
|   |   +-- Config.h
|   |   +-- KeypadInput.h
|   |   +-- LcdDisplay.h
|   |   +-- LedControl.h
|   |   +-- StdioSerial.h
|   +-- src/
|   |   +-- main.cpp
|   |   +-- KeypadInput.cpp
|   |   +-- LcdDisplay.cpp
|   |   +-- LedControl.cpp
|   |   +-- StdioSerial.cpp
+-- Report/
    +-- lab2_report.tex
```

Module responsibilities:

- `KeypadInput`: matrix scan and key retrieval.
- `LcdDisplay`: LCD initialization and message rendering.
- `LedControl`: success/failure LED control.
- `StdioSerial`: UART callbacks and STDIO stream redirection.
- `main.cpp`: application flow and state transitions.

Detailed software module description and interaction:

- `Config.h` **(configuration layer):** centralizes constants used by all modules (pin map, code length, default code, debounce and display timings). This keeps hardware-specific definitions in one place and avoids duplicated literals across source files.
- `KeypadInput.h/.cpp` **(input acquisition layer):** encapsulates the `Keypad` library and matrix mapping. The module exports a single high-level read operation (`getKey()`) and hides low-level row/column scanning details. Its output is a normalized key character (‘0..9‘, ‘*‘, ‘#‘, etc.), consumed by the application state machine.
- `LcdDisplay.h/.cpp` **(visual feedback layer):** wraps the parallel LCD driver and provides a small API (initialize, print two-line messages, clear). This module is responsible only for display rendering and does not contain business logic (validation or mode switching), which preserves separation of concerns.
- `LedControl.h/.cpp` **(status signaling layer):** encapsulates LED GPIO control with semantic operations (valid, invalid, off). The rest of the application never manipulates LED pins directly; it calls intention-level functions, improving readability and maintainability.
- `StdioSerial.h/.cpp` **(communication abstraction layer):** binds C STDIO streams (`stdin`/`stdout`) to UART callbacks through `fdev_setup_stream()`. This enables direct use of `printf`/`scanf` in application code while keeping serial transport details isolated in one module.
- `main.cpp` **(orchestration and state machine):** coordinates all modules and implements system behavior. It maintains input buffers, runtime password, and mode transitions (normal entry, old-code verification, new-code entry), and triggers LCD/LED/serial actions based on events.

Inter-module workflow (mapped to Fig. 1 and Fig. 2):

1. **Initialization phase:** `main.cpp` starts UART, calls `initializeStdioSerial()`, initializes LED and LCD modules, then prints startup guidance via `printf()`.
2. **Input phase:** `main.cpp` polls `KeypadInput::getKey()` and receives one key event at a time.
3. **Processing phase:** the state machine interprets keys (‘*‘ clear, ‘#‘ mode toggle/cancel, digits append), updates buffers, and decides when a full 4-digit code is ready.
4. **Output phase:** based on evaluation result, `main.cpp` calls `LcdDisplay::printMessage()` and `LedControl::showValid()/showInvalid()`, while also logging via `printf()`.

5. **Loop/reset phase:** after status timeout, LEDs are turned off, buffers are reset, and the system returns to waiting for the next key sequence.

This modular interaction pattern ensures that each module has a single clear responsibility and communicates through narrow interfaces, which simplifies debugging, testing, and future extension.

STDIO usage:

- output messages with `printf()` for diagnostics and interaction;
- numeric parsing with `scanf("%d", &numericCode)`;
- stream binding through `fdev_setup_stream()` to `stdin`/`stdout`.

Current pin mapping (latest code version):

- LCD 16x2 (4-bit parallel): `RS=13`, `E=12`, `D4=11`, `D5=10`, `D6=9`, `D7=8`, with `RW` and `VO` tied to `GND`.
- Keypad 4x4: rows `7,6,5,4` and columns `3,2,A3,A2`.
- LEDs: green on `A0` and red on `A1`, each in series with 220 Ω resistor.

Configuration and code excerpts (full source in repository):

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:uno]
platform = atmelavr
board = uno
framework = arduino
lib_deps =
        chris--a/Keypad @ ^3.1.1
        arduino-libraries/LiquidCrystal @ ^1.0.7
```

Listing 1: platformio.ini — project configuration

```
#ifndef CONFIG_H
#define CONFIG_H

```

```
#include <Arduino.h>

#define SerialBaudRate 9600

#define GreenLedPin A0
#define RedLedPin A1

#define KeypadRows 4
#define KeypadCols 4

static const byte KeypadRowPins[KeypadRows] = {7, 6, 5, 4};
static const byte KeypadColPins[KeypadCols] = {3, 2, A3, A2};

#define CodeLength 4
#define ValidAccessCode "1234"

#define LcdColumns 16
#define LcdRows 2

#define LcdRsPin 13
#define LcdEnPin 12
#define LcdD4Pin 11
#define LcdD5Pin 10
#define LcdD6Pin 9
#define LcdD7Pin 8

#define KeyDebounceDelayMs 180
#define StatusDisplayDelayMs 2000

#endif
```

Listing 2: Config.h — pins, constants, defaults

```
#ifndef LED_CONTROL_H
#define LED_CONTROL_H

#include <Arduino.h>

class LedControl {
public:
    LedControl(uint8_t greenPin, uint8_t redPin);

    void initialize();
    void showValid();
    void showInvalid();
    void turnOff();

```

```
15 private:
16     uint8_t greenLedPin;
17     uint8_t redLedPin;
18 };
19
20 #endif
```

Listing 3: LedControl.h — LED module API

```
1  #ifndef LCD_DISPLAY_H
2  #define LCD_DISPLAY_H
3
4  #include <LiquidCrystal.h>
5
6  class LcdDisplay {
7  public:
8      LcdDisplay();
9
10     void initialize();
11     void printMessage(const char* line1, const char* line2);
12     void clear();
13
14 private:
15     LiquidCrystal lcd;
16 };
17
18 #endif
```

Listing 4: LcdDisplay.h — LCD module API

```
1  #ifndef KEYPAD_INPUT_H
2  #define KEYPAD_INPUT_H
3
4  #include <Keypad.h>
5
6  class KeypadInput {
7  public:
8      KeypadInput();
9      ~KeypadInput();
10
11     char getKey();
12
13 private:
14     Keypad* keypad;
15 };
16
17 #endif
```

Listing 5: KeypadInput.h — keypad module API

```
#ifndef STDIO_SERIAL_H
#define STDIO_SERIAL_H

void initializeStdioSerial();

#endif
```

Listing 6: StdioSerial.h — STDIO serial API

```
#include "LedControl.h"

LedControl::LedControl(uint8_t greenPin, uint8_t redPin)
    : greenLedPin(greenPin), redLedPin(redPin) {}

void LedControl::initialize() {
    pinMode(greenLedPin, OUTPUT);
    pinMode(redLedPin, OUTPUT);
    turnOff();
}

void LedControl::showValid() {
    digitalWrite(greenLedPin, HIGH);
    digitalWrite(redLedPin, LOW);
}

void LedControl::showInvalid() {
    digitalWrite(greenLedPin, LOW);
    digitalWrite(redLedPin, HIGH);
}

void LedControl::turnOff() {
    digitalWrite(greenLedPin, LOW);
    digitalWrite(redLedPin, LOW);
}
```

Listing 7: LedControl.cpp — LED logic

```
#include "LcdDisplay.h"

#include "Config.h"

LcdDisplay::LcdDisplay() : lcd(LcdRsPin, LcdEnPin, LcdD4Pin, LcdD5Pin,
    LcdD6Pin, LcdD7Pin) {}

```

```
void LcdDisplay::initialize() {
    lcd.begin(LcdColumns, LcdRows);
}

void LcdDisplay::printMessage(const char* line1, const char* line2) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(line1);
    lcd.setCursor(0, 1);
    lcd.print(line2);
}

void LcdDisplay::clear() {
    lcd.clear();
}
```

Listing 8: LcdDisplay.cpp — LCD behavior

```
#include "KeypadInput.h"

#include "Config.h"

char keys[KeypadRows][KeypadCols] = {
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};

KeypadInput::KeypadInput() {
    keypad = new Keypad(makeKeymap(keys), KeypadRowPins, KeypadColPins,
        KeypadRows, KeypadCols);
}

KeypadInput::~KeypadInput() {
    delete keypad;
}

char KeypadInput::getKey() {
    return keypad->getKey();
}
```

Listing 9: KeypadInput.cpp — keypad mapping and read

```
#include <Arduino.h>
#include <stdio.h>

```

```
#include "StdioSerial.h"

namespace {
FILE serialOutputStream;
FILE serialInputStream;

int serialPutChar(char character, FILE*) {
    if (character == '\n') {
        Serial.write('\r');
    }

    Serial.write(character);
    return 0;
}

int serialGetChar(FILE*) {
    while (!Serial.available()) {
    }

    return Serial.read();
}
} // namespace

void initializeStdioSerial() {
    fdev_setup_stream(&serialOutputStream, serialPutChar, nullptr,
        _FDEV_SETUP_WRITE);
    stdout = &serialOutputStream;

    fdev_setup_stream(&serialInputStream, nullptr, serialGetChar,
        _FDEV_SETUP_READ);
    stdin = &serialInputStream;
}
```

Listing 10: StdioSerial.cpp — STDIO/UART glue code

```
#include <Arduino.h>
#include <stdio.h>
#include <string.h>

#include "Config.h"
#include "LedControl.h"
#include "LcdDisplay.h"
#include "KeypadInput.h"
#include "StdioSerial.h"

LedControl ledControl(GreenLedPin, RedLedPin);
LcdDisplay lcdDisplay;
```

```
13 KeypadInput keypadInput;
14
15 char enteredCode[CodeLength + 1];
16 char runtimeAccessCode[CodeLength + 1];
17 uint8_t currentIndex = 0;
18
19 enum class InputMode {
20     EnterAccessCode,
21     VerifyCurrentCode,
22     EnterNewCode
23 };
24
25 InputMode inputMode = InputMode::EnterAccessCode;
26
27 void clearInputBuffer() {
28     memset(enteredCode, 0, sizeof(enteredCode));
29     currentIndex = 0;
30 }
31
32 void showPromptForMode() {
33     if (inputMode == InputMode::EnterAccessCode) {
34         lcdDisplay.printMessage("Enter 4-digit", "code:");
35         return;
36     }
37
38     if (inputMode == InputMode::VerifyCurrentCode) {
39         lcdDisplay.printMessage("Old code:", "");
40         return;
41     }
42
43     lcdDisplay.printMessage("New code:", "");
44 }
45
46 void resetCodeInput() {
47     clearInputBuffer();
48     showPromptForMode();
49 }
50
51 void updateCodePreview() {
52     char preview[CodeLength + 1];
53
54     for (uint8_t index = 0; index < currentIndex; index++) {
55         preview[index] = '*';
56     }
57
58     preview[currentIndex] = '\0';
```

```
59      if (inputMode == InputMode::EnterAccessCode) {
60          lcdDisplay.printMessage("Enter 4-digit", preview);
61          return;
62      }
63
64      if (inputMode == InputMode::VerifyCurrentCode) {
65          lcdDisplay.printMessage("Old code:", preview);
66          return;
67      }
68
69      lcdDisplay.printMessage("New code:", preview);
70  }
71
72  void evaluateAccessCode() {
73      enteredCode[currentIndex] = '\0';
74
75      int numericCode = 0;
76      int convertedValues = 0;
77      if (Serial.available() > 0) {
78          convertedValues = scanf("%d", &numericCode);
79      }
80      bool isValidCode = strcmp(enteredCode, runtimeAccessCode) == 0;
81
82      printf("Entered Code: %s\n", enteredCode);
83      printf("Scanf parse status: %d\n", convertedValues);
84      printf("Scanf parsed value: %d\n", numericCode);
85
86      if (isValidCode) {
87          printf("Access Granted\n");
88          lcdDisplay.printMessage("Access", "Granted");
89          ledControl.showValid();
90      } else {
91          printf("Access Denied\n");
92          lcdDisplay.printMessage("Access", "Denied");
93          ledControl.showInvalid();
94      }
95
96      delay(StatusDisplayDelayMs);
97      ledControl.turnOff();
98      resetCodeInput();
99  }
100
101 void evaluateCurrentCodeForChange() {
102     enteredCode[currentIndex] = '\0';
103     printf("Verify old code: %s\n", enteredCode);
104
```

```
105     if (strcmp(enteredCode, runtimeAccessCode) == 0) {
106         printf("Old code correct. Enter new code.\n");
107         inputMode = InputMode::EnterNewCode;
108         resetCodeInput();
109         return;
110     }
111 
112     printf("Old code invalid. Password unchanged.\n");
113     lcdDisplay.printMessage("Old code", "invalid");
114     ledControl.showInvalid();
115     delay(StatusDisplayDelayMs);
116     ledControl.turnOff();
117     inputMode = InputMode::EnterAccessCode;
118     resetCodeInput();
119 }
120 
121 void saveNewCode() {
122     enteredCode[currentIndex] = '\0';
123     strcpy(runtimeAccessCode, enteredCode);
124 
125     printf("Password changed successfully.\n");
126     lcdDisplay.printMessage("Password", "Updated");
127     ledControl.showValid();
128     delay(StatusDisplayDelayMs);
129     ledControl.turnOff();
130 
131     inputMode = InputMode::EnterAccessCode;
132     resetCodeInput();
133 }
134 
135 void evaluateCodeByMode() {
136     if (inputMode == InputMode::EnterAccessCode) {
137         evaluateAccessCode();
138         return;
139     }
140 
141     if (inputMode == InputMode::VerifyCurrentCode) {
142         evaluateCurrentCodeForChange();
143         return;
144     }
145 
146     saveNewCode();
147 }
148 
149 void setup() {
150     Serial.begin(SerialBaudRate);
```

```
151     initializeStdioSerial();
152 
153     strcpy(runtimeAccessCode, ValidAccessCode);
154 
155     ledControl.initialize();
156     lcdDisplay.initialize();
157 
158     resetCodeInput();
159     printf("System Ready. Enter Code via Keypad.\n");
160     printf("Press # to change password, * to clear input.\n");
161 }
162 
163 void loop() {
164     char key = keypadInput.getKey();
165 
166     if (key != '\0') {
167         printf("Key Pressed: %c\n", key);
168 
169         if (key == '#') {
170             if (inputMode == InputMode::EnterAccessCode && currentIndex
                    == 0) {
171                 inputMode = InputMode::VerifyCurrentCode;
172                 printf("Password change mode enabled.\n");
173                 resetCodeInput();
174                 delay(KeyDebounceDelayMs);
175                 return;
176             }
177 
178             if (inputMode != InputMode::EnterAccessCode) {
179                 inputMode = InputMode::EnterAccessCode;
180                 printf("Password change cancelled.\n");
181                 resetCodeInput();
182                 delay(KeyDebounceDelayMs);
183                 return;
184             }
185         }
186 
187         if (key == '*') {
188             printf("Input cleared\n");
189             resetCodeInput();
190             delay(KeyDebounceDelayMs);
191             return;
192         }
193 
194         if (key >= '0' && key <= '9' && currentIndex < CodeLength) {
195             enteredCode[currentIndex++] = key;
```

```
            updateCodePreview();

            if (currentIndex == CodeLength) {
                evaluateCodeByMode();
            }
        }

        delay(KeyDebounceDelayMs);
    }
}
```

Listing 11: main.cpp — main application flow

Results

The firmware builds successfully and demonstrates required behavior in simulation/board execution.

```
> pio run
Processing uno (platform: atmelavr; board: uno; framework: arduino)
...
RAM:   38.5% (789/2048 bytes)
Flash: 29.6% (9562/32256 bytes)
========================== [SUCCESS] ==========================
```

Listing 12: PlatformIO build summary

Functional scenarios confirmed:

1. Enter correct 4-digit code → LCD “Access Granted” and green LED ON.
2. Enter wrong code → LCD “Access Denied” and red LED ON.
3. Press * → clear current input buffer.
4. Press # at empty input → enter/cancel password change mode.

```
System Ready. Enter Code via Keypad.
Press # to change password, * to clear input.
Key Pressed: 1
Key Pressed: 2
Key Pressed: 3
Key Pressed: 4
Entered Code: 1234
Scanf parse status: 0
Scanf parsed value: 0
Access Granted
```

Listing 13: Example STDIO serial session

Visual evidence:

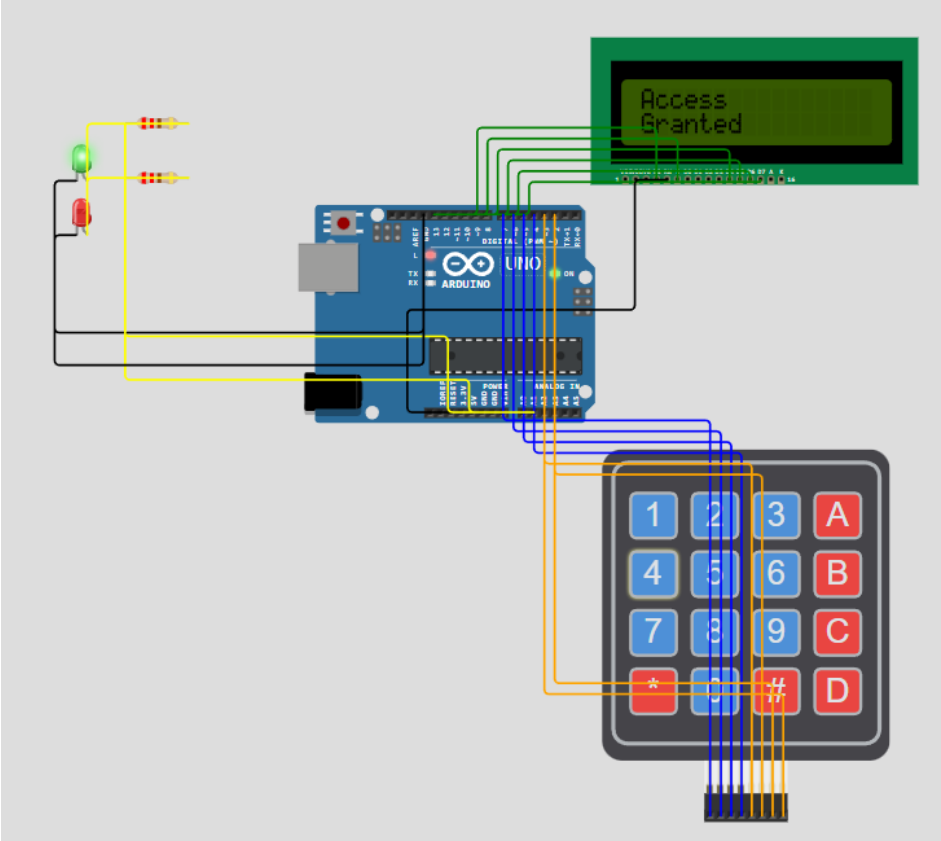

Figure 3: Simulation screenshot with keypad, LCD and LEDs

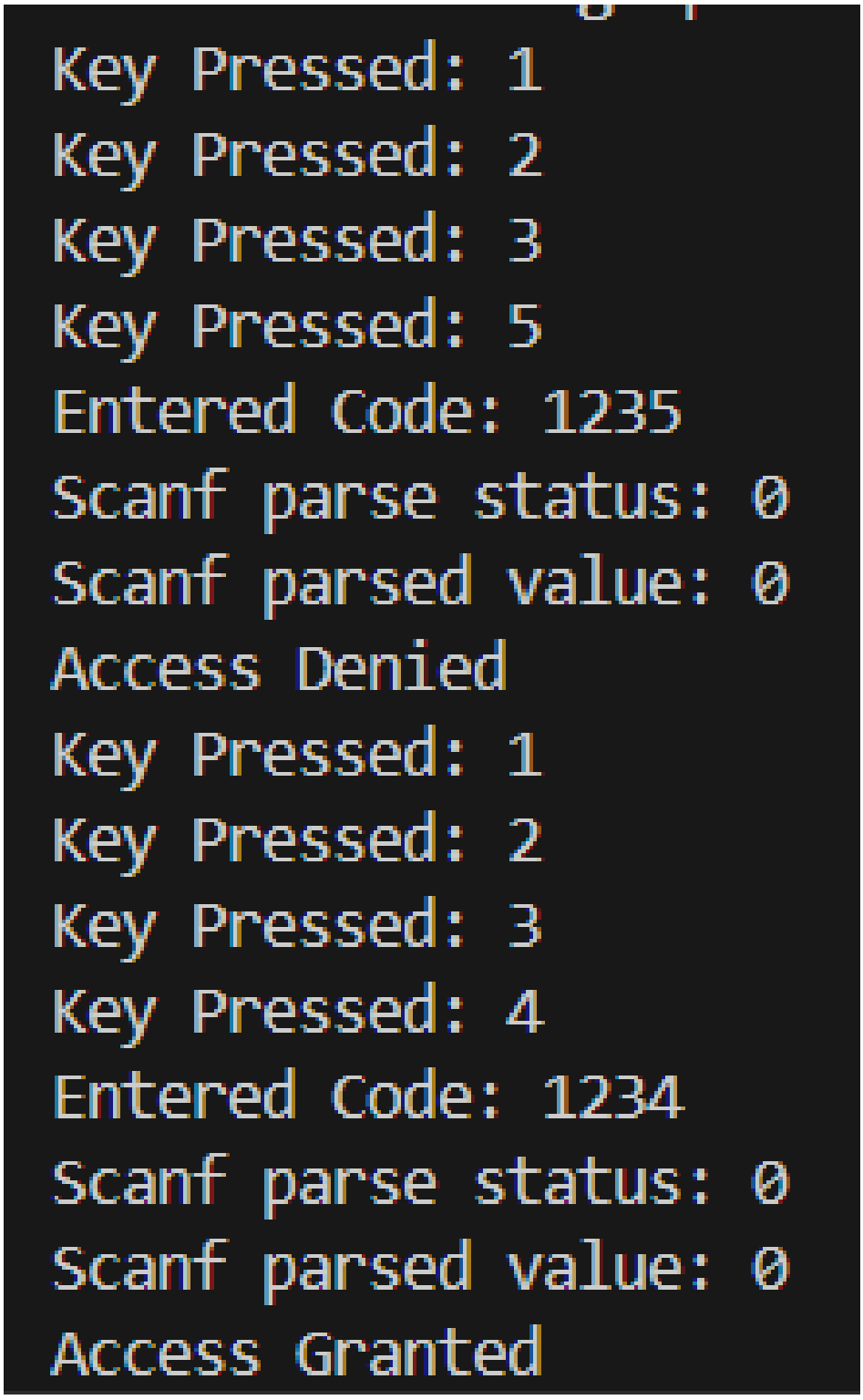

Figure 4: Serial monitor output proving STDIO interaction

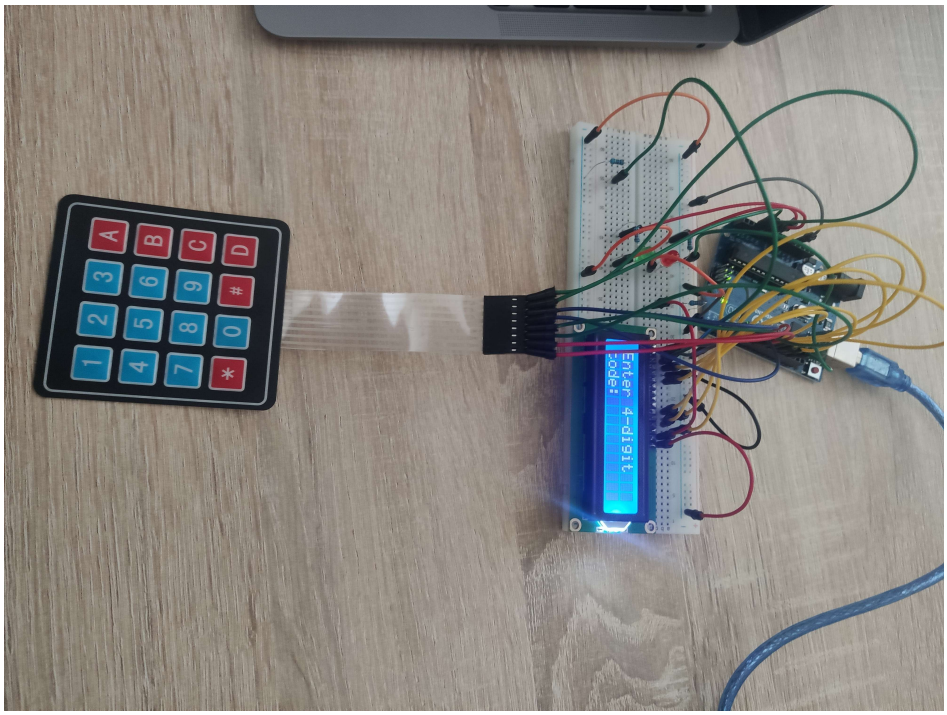

Figure 5: Physical setup photo (if hardware implementation is used)

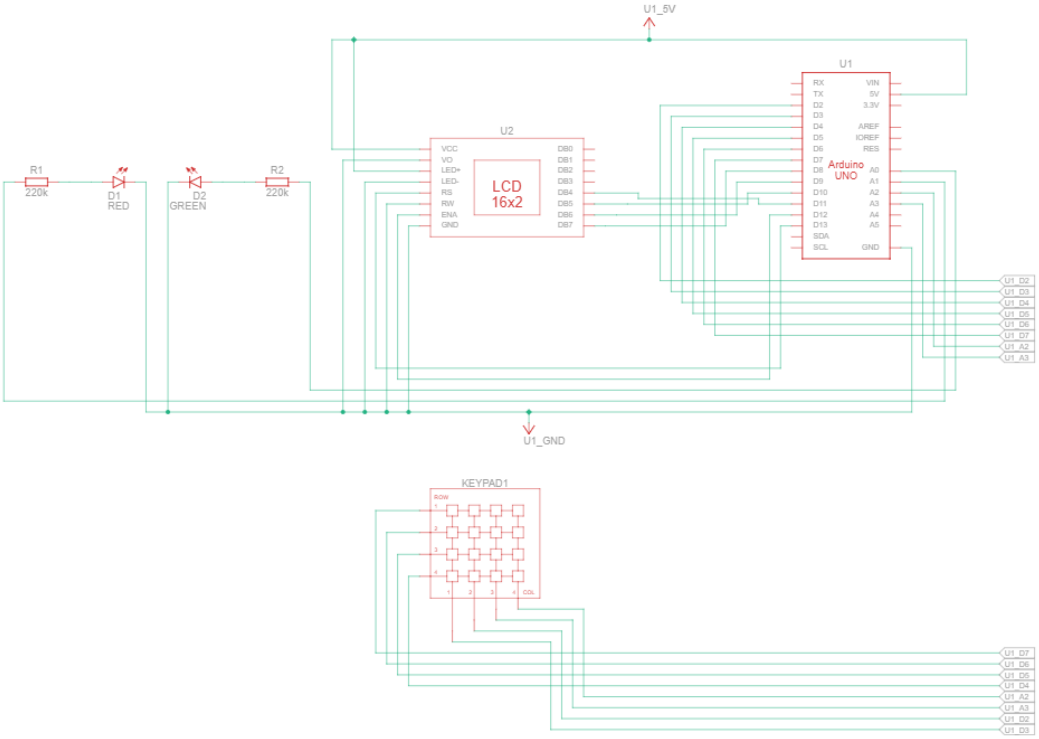

Figure 6: Electrical circuit diagram

Checklist coverage summary:

Table 2: Checklist compliance matrix

Checklist Item	**Covered**	**Evidence**
Analiza tehnologică și context	Yes	Theory Background
Proiectare arhitecturală + explicații HW–SW	Yes	Figs. 1, 2
Structură modulară (.h/.cpp)	Yes	Technical implementation
Descriere componente hardware	Yes	Components table
Descriere module software	Yes	Module responsibilities + listings
Implementare funcțională	Yes	Functional scenarios
Utilizare STDIO (`printf`, `scanf`)	Yes	STDIO usage + serial log
Dovezi rezultate (capturi/foto)	Yes	Visual evidence section
Raport complet (cercetare/proiectare/rezultate + concluzii + bibliografie)	Yes	All report sections
Notă AI + anexă cod sursă	Yes	Final sections
Funcționalitate suplimentară (opțional)	Yes	Password change mode

Conclusion

This laboratory work successfully achieved all objectives set out in the assignment. The developed system enables secure user interaction with an embedded platform using a 4x4 keypad for input, a 16x2 parallel LCD for feedback, and green/red LEDs (through 220Ω series resistors) for immediate status indication. The use of STDIO functions (`printf`, `scanf`) for serial communication demonstrates a robust and portable approach to text-based I/O, fulfilling both technical and educational requirements.

The project was implemented with a modular architecture, separating hardware abstraction, logic, and user interface into distinct, well-documented modules. This structure not only improves maintainability and scalability but also aligns with best practices in embedded software engineering.

Key achievements include:

- Reliable keypad code entry and validation with clear LCD and LED feedback.
- Correct and efficient use of STDIO for serial communication, including stream redirection.
- Implementation of an optional password change mode, enhancing usability and demonstrating extensibility.
- Comprehensive documentation and code organization, facilitating future development and code reuse.

Challenges encountered included ensuring debounced keypad input, synchronizing LCD updates with user actions, and integrating STDIO with the Arduino serial interface. These were overcome through careful design, testing, and iterative refinement.

Lessons learned:

- Modular design greatly simplifies debugging and future feature additions.
- Using standard C I/O (STDIO) on microcontrollers, while non-trivial, provides significant portability and code clarity benefits.
- Clear user feedback (via LCD and LEDs) is essential for usability in embedded systems.

Possible future improvements:

- Add persistent storage (EEPROM) for password retention after power loss.
- Implement unit tests for all modules to ensure reliability.
- Optimize input handling with interrupt-driven routines for even better responsiveness.
- Expand the user interface with additional features, such as lockout after repeated failed attempts.

Overall, this project provided valuable hands-on experience in embedded system design, modular programming, and user interface development, and serves as a solid foundation for more advanced applications.

Bibliography

1. ATmega328P Datasheet, Microchip. `https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega328P-Data-Sheet-DS40002061B.pdf`

2. AVR Libc User Manual — STDIO Facilities. https://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html

3. PlatformIO Documentation (Arduino Uno). https://docs.platformio.org/en/latest/boards/atmelavr/uno.html

4. Arduino Language Reference — Serial Communication. https://www.arduino.cc/reference/en/language/functions/communication/serial/

5. GitHub Copilot Documentation. https://docs.github.com/en/copilot

Note on AI Tool Usage

This report and parts of the project source code were prepared with assistance from GitHub Copilot (AI). All AI-generated text and code suggestions were reviewed, validated, tested, and adapted by the author before final inclusion.

Appendix

GitHub repository link: https://github.com/Tirppy/si-course-repo/tree/main/Lab2/Code